# ΜΥΥ802 - Μεταφραστές

# Προγραμματιστική άσκηση:

# Η γλώσσα προγραμματισμού greek++

Χαρίλαος Χατζηδημητρίου – 5387

Όμηρος Χατζηϊορδάνης – 5388

# 1. Εισαγωγή στον Μεταγλωττιστή της Γλώσσας greek++

## 1.1 Ο Ρόλος του Μεταγλωττιστή

Ο μεταγλωττιστής (**compiler**) είναι ένα θεμελιώδες εργαλείο στην πληροφορική, το οποίο μετατρέπει τον πηγαίο κώδικα (**source code**) μιας γλώσσας προγραμματισμού σε ισοδύναμη έκφραση σε μια άλλη γλώσσα (συνήθως σε γλώσσα μηχανής ή Assembly). Στην περίπτωση της γλώσσας greek++, ο μεταγλωττιστής αναλαμβάνει τη μετάφραση προγραμμάτων γραμμένων σε αυτήν την γλώσσα σε εκτελέσιμο κώδικα.

## 1.2 Δομή και Στάδια Ανάπτυξης

Η ανάπτυξη του μεταγλωττιστή χωρίζεται σε **δύο κύρια μέρη**:

**Front-end**

Αποτελεί το πρώτο στάδιο επεξεργασίας και περιλαμβάνει:

1. **Λεκτικός Αναλυτής (Lexer)** – Διαχωρίζει τον κώδικα σε tokens (λέξεις-κλειδιά, αναγνωριστικά, τελεστές).
2. **Συντακτικός Αναλυτής (Parser)** – Ελέγχει τη σύνταξη (syntax) του κώδικα βάσει γραμματικών κανόνων.
3. **Σημασιολογική Ανάλυση** – Επαληθεύει τη σημασιολογική ορθότητα (π.χ. δηλώσεις).
4. **Παραγωγή Ενδιάμεσου Κώδικα** – Δημιουργία μιας ενδιάμεσης αναπαράστασης για βελτιστοποίηση.

**Back-end**

Ασχολείται με τη δημιουργία του τελικού εκτελέσιμου κώδικα:
5. **Πίνακας Συμβόλων (Symbol Table)** – Καταγράφει μεταβλητές, συναρτήσεις και άλλες πληροφορίες.
6. **Παραγωγή Τελικού Κώδικα** – Μετατροπή του ενδιάμεσου κώδικα σε κώδικα μηχανής ή Assembly.

## 1.3. Σκοπός της Αναφοράς

Ο σκοπός αυτής της αναφοράς είναι να παρουσιάσει τη γλώσσα προγραμματισμού **greek++**, μια εκπαιδευτική γλώσσα που αναπτύχθηκε στο πλαίσιο του μαθήματος «**Μεταφραστές**» (**ΜΥΥ802**) στο Τμήμα Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων. Η αναφορά περιλαμβάνει τη λεπτομερή περιγραφή της γλώσσας, τις βασικές της δομές, τους κανόνες σύνταξης και τη γραμματική της, καθώς και παραδείγματα προγραμμάτων.

## 1.4. Ποια είναι η γλώσσα greek++;

Η γλώσσα προγραμματισμού greek++ είναι μια εκπαιδευτική, ελληνόφωνη γλώσσα που παρά τον σχετικά απλό και περιορισμένο της σχεδιασμό, η κατασκευή του μεταγλωττιστή για τη greek++ παρουσιάζει ιδιαίτερο ενδιαφέρον, καθώς περιλαμβάνει βασικά χαρακτηριστικά που απαντώνται σε πληρέστερες γλώσσες προγραμματισμού.

## 1.5. Βασικά Χαρακτηριστικά της greek++

Η γλώσσα χρησιμοποιεί ελληνικές και αγγλικές λέξεις-κλειδιά, καθώς και σύμβολα για τελεστές και διαχωριστές. Οι δεσμευμένες λέξεις της γλώσσας (π.χ., εάν, τότε, για, συνάρτηση) δεν μπορούν να χρησιμοποιηθούν ως αναγνωριστικά μεταβλητών.

Η greek++ υποστηρίζει:

- **βασικούς τύπους δεδομένων**, όπως ακέραιους και πραγματικούς αριθμούς,
- **εντολές ελέγχου ροής** (όπως εάν, όσο, επανάλαβε, για),
- **δηλώσεις μεταβλητών**,
- **εισόδους/εξόδους** (διάβασε, γράψε),
- και **υποπρογράμματα** με **παραμέτρους** (διαδικασίες και συναρτήσεις).

Επιπλέον, περιλαμβάνει δυνατότητες για **μετάδοση παραμέτρων με τιμή ή αναφορά**, ενώ ενσωματώνει έννοιες όπως **αναδρομή**, κάτι που την καθιστά ιδανική για διδακτική προσέγγιση θεμάτων όπως ανάλυση συντακτικού, σημασιολογικός έλεγχος και παραγωγή ενδιάμεσου κώδικα.

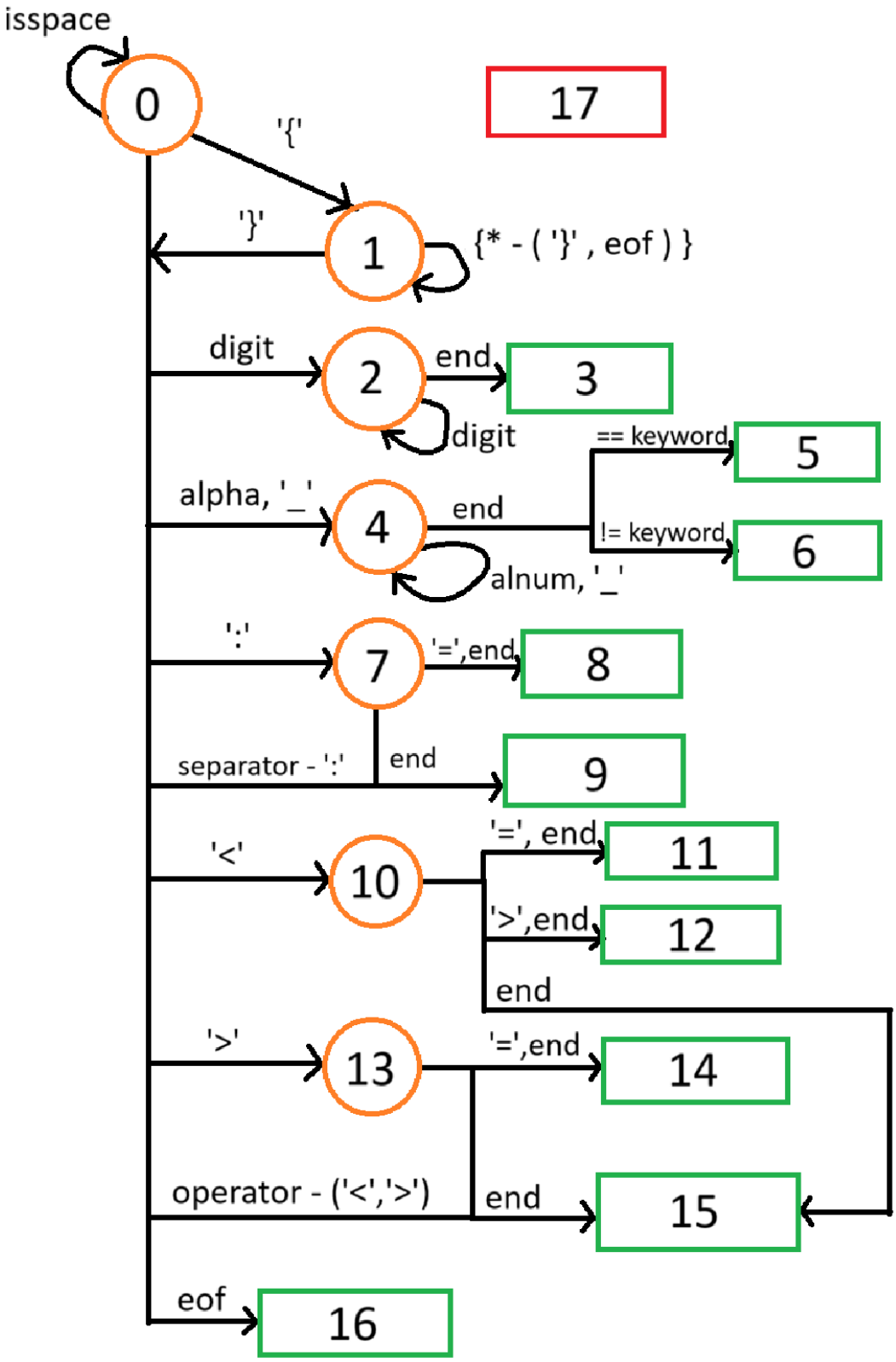
# 2. Λεκτικός Αναλυτής

Ο λεκτικός αναλυτής υλοποιείται από τη κλάση **Lexer**. Η κλάση περιέχει όλο το κείμενο του αρχείου **.gr**, τις δεσμευμένες λέξεις της γλώσσας, τα σύμβολα των αριθμητικών πράξεων, τους τελεστές συσχέτισης, την τωρινή τοποθεσία σε γραμμές και στήλες και τις συναρτήσεις για την ανάλυση της κάθε λέξης του κειμένου.

Επίσης υπάρχει η κλάση **Token** που αντιπροσωπεύει τη κάθε λέξη ή σύμβολο στο πρόγραμμα. Στη κλάση περιέχονται ο τύπος του token που περιγράφεται από τη κλάση **TokenType**, τη τιμή του token, και τέλος τη γραμμή και τη στήλη του που βοηθούν στην τύπωση σφαλμάτων στο πρόγραμμα.

## 2.1. Διάγραμμα Καταστάσεων

Ο λεκτικός αναλυτής μπορεί να περιγράφει από το αυτόματο που φαίνεται στην παρακάτω εικόνα.



## 2.2. Περιγραφή Καταστάσεων Αυτομάτου

Οι καταστάσεις του αναλυτή είναι:

- **Κατάσταση 0:** Η αρχική κατάσταση του αναλυτή όταν ξεκινάει, και αναλυεί το κάθε γράμμα του token. Άμα βρεθεί κενός χαρακτήρας, παραμένει σε αυτήν την κατάσταση.
- **Κατάσταση 1:** Ο αναλυτής έχει εντοπίσει το σύμβολο {, οπότε προκύπτει σχόλιο και άρα αγνοούνται όλα τα σύμβολα που υπάρχουν μέσα και παραμένει στη κατάσταση αυτή, μέχρι να βρεθεί το σύμβολο } και άρα να τελειώσει το σχόλιο.
- **Κατάσταση 2:** Ο αναλυτής βρίσκει ψηφίο, όσο βρίσκει ψηφίο θα συνεχίσει να βρίσκεται σε αυτή την κατάσταση.
- **Κατάσταση 3:** Ο αναλυτής βρίσκει ακέραιο αριθμό και τον τοποθετεί στα tokens.
- **Κατάσταση 4**: Αλφαβητικό σύμβολο ή τον χαρακτήρα **'_'** άρα πρόκειται είτε για όνομα μεταβλητής είτε για κάποια δεσμευμένη λέξη. Ο ανάλυτης παραμένει στην ίδια κατάσταση με οποιοδήποτε αλφαρηθμιτικό σύμβολο έρθει και στο τέλος ελέγχει αν το token αποτελεί δεσμευμένη λέξη ή όχι και πάει στην αντίστοιχη κατάσταση.
- **Κατάσταση 5**: Ο αναλυτής εντόπισε δεσμευμένη λέξη και την τοποθετεί.
- **Κατάσταση 6**: Ο αναλυτής δεν εντόπισε δεσμευμένη λέξη, άρα τοποθετεί το token ως identifier.
- **Κατάσταση 7**: Ο αναλυτής βρίσκει το σύμβολο :, αν το επόμενο σύμβολο είναι το = πηγαίνει στη κατάσταση 8, αλλιώς πηγαίνει στην 9.
- **Κατάσταση 8**: Ο αναλυτής βρήκε τον **operator :=** και τον τοποθετεί στα tokens.
- **Κατάσταση 9**: Ο αναλυτής βρίσκει separator και τον τοποθετεί στα tokens.
- **Κατάσταση 10**: Ο αναλυτής βρίσκει το σύμβολο **<** και ελέγχει αν το επόμενο σύμβολο είναι το **'='** για να πάει στην **κατάσταση 11**, αλλιώς αν βρει το σύμβολο **'>'** πηγαίνει στην **κατάσταση 12**, και αν δεν βρει τίποτα από τα άλλα δύο πηγαίνει στην **κατάσταση 15**.
- **Κατάσταση 11**: Ο αναλυτής βρει τον operator **<=**, και τον τοποθετεί στα tokens.
- **Κατάσταση 12**: Ο αναλυτής βρήκε τον operator **<>** και τον τοποθετεί στα tokens.
- **Κατάσταση 13:** Ο αναλυτής βρίσκει το σύμβολο **>** και ελέγχει αν το επόμενο σύμβολο είναι το **'='** για να πάει στην **κατάσταση 14**, αλλιώς πάει στην **κατάσταση 15**.
- **Κατάσταση 14**: Ο αναλυτής βρήκε τον **operator >=** και τον τοποθετεί στα tokens.

- **Κατάσταση 15**: Ο αναλυτής βρήκε έναν από τους operator που δεν υπάρχουν στις καταστάσεις **8, 11, 12 και 14**.
- **Κατάσταση 16**: Ο αναλυτής βρίσκει το τέλος του αρχείου και τελειώνει η ανάλυση του.
- **Κατάσταση 17**: Ο αναλυτής έχει βρει σύμβολο που δεν ταιριάζει σε κάποια κατάσταση και μπαίνει σε κατάσταση σφάλματος, βγάζει μήνυμα σφάλματος και σταματάει την ανάλυση.

Το end στο αυτόματο σημαίνει ότι τελειώνει το token.

Σε κάθε κατάσταση αν δεν αναφέρεται κάποιο σύμβολο, τότε οδηγεί στην κατάσταση 17.

Οι παραπάνω διαδικασίες υλοποιούνται από τις συναρτήσεις του Lexer.

Αρχικά από την συνάρτηση tokenize η οποία ελέγχει αν το σύμβολο που ελέγχεται είναι κενό ή όχι. Αν δεν είναι καλεί την get_next_token ώστε να λάβει το token που προκύπτει με το σωστό τύπο και να το τοποθετήσει σε έναν πίνακα τον οποίο θα επιστρέψει όταν τελειώσει όλο το κείμενο του προγράμματος.

Η get_next_token ελέγχει το πρώτο σύμβολο που της έρχεται και ανάλογα με τι είναι, καλεί μια από τις υπόλοιπες συναρτήσεις που ταιριάζει.

Οι συναρτήσεις advance και peek είναι βοηθητικές για τις υπόλοιπες. Η πρώτη είναι υπεύθυνη για τη σωστή αλλαγή στο επόμενο σύμβολο του κειμένου, αλλάζοντας και κατάλληλα της γραμμές και τις στήλες που κρατάει ο αναλυτής. Η peek είναι χρήσιμη ώστε να ελέγχονται τα πιθανά επόμενα σύμβολα και να μπαίνει ο αναλυτής στη σωστή κατάσταση, χωρίς να καταναλώνεται το σύμβολο που χρησιμοποιείται ήδη.

Τέλος οι συναρτήσεις skip_whitespace, skip_comment, get_integer_token, get_identifier_or_keyword_token, get_operator_token, get_separator_token αντιπροσωπεύουν τις καταστάσεις οι οποίες τοποθετούν tokens με κατάλληλο τύπο και καλούν την advance και peek άμα χρειαστεί. Συγκεκριμένα οι συναρτήσεις skip απλά προσπερνάνε χαρακτήρες, με τον αναλυτή να είναι σε αντίστοιχες καταστάσεις που επιγράφτηκαν πάνω, και δεν τοποθετούν νέα token. Οι υπόλοιπες κάνουν ελέγχους μέχρι να βρουν ότι η λέξη τους τέλειωσε, όπου την επιστρέφουν στην get_next_token, και κατά συνέπεια στην tokenize, με τον κατάλληλο τύπο και τοποθεσία για τυχόν σφάλματα που μπορεί να προκύψουν σε επόμενα κομμάτια του μεταφραστή.

## 2.3. Γενική Περιγραφή της Κλάσης Lexer

Η κλάση **Lexer** αποτελεί τον λεκτικό αναλυτή του μεταγλωττιστή. Ο ρόλος της είναι να διαβάζει το πηγαίο κείμενο του προγράμματος χαρακτήρα-χαρακτήρα και να το

μετατρέπει σε μια ακολουθία από λεκτικές μονάδες (**tokens**), οι οποίες χρησιμοποιούνται από τα επόμενα στάδια της μεταγλώττισης (**parser**).

### 2.3.1. Κύρια χαρακτηριστικά

- **Αρχικοποίηση:** Δέχεται ως είσοδο το πλήρες κείμενο του προγράμματος και αρχικοποιεί δείκτες για τη θέση, τη γραμμή και τη στήλη.
- **Αναγνώριση tokens:** Αναγνωρίζει keywords, identifiers, ακέραιους αριθμούς, τελεστές, διαχωριστικά και το τέλος αρχείου.
- **Υποστήριξη σχολίων και κενών:** Παραλείπει σχόλια και κενά διαστήματα.
- **Έλεγχος λαθών:** Εντοπίζει μη επιτρεπτούς χαρακτήρες και υπερβολικά μεγάλα **identifiers** (μεγαλύτερα των 30 char).
- **Υποστήριξη ελληνικών:** Λειτουργεί με ελληνικούς χαρακτήρες και keywords.
- **Βασικές μέθοδοι:**
    - **advance()**: Μετακινεί τον δείκτη στον επόμενο χαρακτήρα.
    - **skip_whitespace()**, **skip_comment()**: Παραλείπουν κενά και σχόλια.
    - **get_next_token()**: Επιστρέφει το επόμενο token.
    - **tokenize()**: Επιστρέφει τη λίστα όλων των tokens του προγράμματος.

## 2.4. Λειτουργικότητα και Μέθοδοι

### 2.4.1. Αρχικοποίηση και Βασικές Λειτουργίες

- **Constructor (__init__):** Αρχικοποιεί τον λεκτικό αναλυτή με το κείμενο του προγράμματος, θέτει την αρχική θέση, γραμμή και στήλη, και ορίζει τις λίστες με τις δεσμευμένες λέξεις, τελεστές και διαχωριστικά.
    - **text**: Το πηγαίο κείμενο του προγράμματος.
    - **pos**, **line**, **column**: Τρέχουσα θέση, γραμμή και στήλη.
    - **current_char**: Ο τρέχων χαρακτήρας που αναλύεται.
    - **keywords**, **operators**, **separators**: Σύνολα με τις δεσμευμένες λέξεις, τελεστές και διαχωριστικά της γλώσσας.

### 2.4.2. Ανάλυση και Διαχείριση Token

- **advance()**: Μετακινεί τον δείκτη στον επόμενο χαρακτήρα του κειμένου και ενημερώνει τη θέση, γραμμή και στήλη. Αν δεν υπάρχουν άλλοι χαρακτήρες, θέτει το **current_char** σε **None**.
- **skip_whitespace()**: Παραλείπει τα κενά διαστήματα και τους χαρακτήρες νέας γραμμής, ενημερώνοντας τη γραμμή και στήλη κατάλληλα.

- **skip_comment()**: Αγνοεί τα σχόλια που περικλείονται σε **{...}**, μετακινώντας τον δείκτη μέχρι το τέλος του σχολίου.
- **get_integer_token()**: Αναγνωρίζει και επιστρέφει ένα **token** ακέραιου αριθμού, συλλέγοντας διαδοχικά ψηφία.
- **get_identifier_or_keyword_token()**: Αναγνωρίζει αναγνωριστικά ή δεσμευμένες λέξεις, επιβάλλοντας όριο 30 χαρακτήρων. Ελέγχει αν η λέξη ανήκει στα keywords και επιστρέφει το αντίστοιχο **token**.
- **get_operator_token()**: Αναγνωρίζει τελεστές, συμπεριλαμβανομένων τελεστών όπως {**<=, >=, <>**}.
- **get_separator_token()**: Αναγνωρίζει και επιστρέφει **tokens** για διαχωριστικά όπως **{";", ",", "(", ")", "[", "]", "{", "}", ":"}.**
- **get_next_token()**: Επιστρέφει το επόμενο token από το κείμενο, καλώντας τις κατάλληλες μεθόδους ανάλογα με τον τύπο του χαρακτήρα.
    - Αντιμετωπίζει σχόλια, κενά, αριθμούς, αναγνωριστικά, τελεστές και διαχωριστικά.
    - Εμφανίζει σφάλματα για μη έγκυρους χαρακτήρες.
- **peek()**: Επιστρέφει τον επόμενο χαρακτήρα χωρίς να μετακινήσει τον δείκτη.
- **tokenize()**: Διαχωρίζει όλο το κείμενο σε μια λίστα **tokens**, καλώντας επανειλημμένα τη **get_next_token()**.

### 2.4.3. Βοηθητική Κλάση

- Token **Class:** Αποθηκεύει πληροφορίες για κάθε token, όπως τύπος (**TokenType**), τιμή, γραμμή και στήλη. Αυτό μας είναι πολύ χρήσιμο στην λειτουργία debugging του μεταγλωττιστή μας, καθώς μας εμφανίζει τις απαραίτητες πληροφορίες για το κάθε Token και συνεπώς διευκολύνει τον εντοπισμό σφαλμάτων.
    - **__str__ και __repr__**: Επιστρέφουν αναπαράσταση του token για **debugging**.

### 2.4.4. Διαχείριση Σφαλμάτων

- **Έλεγχος μη έγκυρων χαρακτήρων:** Εμφανίζει σφάλματα με λεπτομέρειες (γραμμή, στήλη) για μη αναγνωρίσιμους χαρακτήρες.
- **Έλεγχος μήκους αναγνωριστικών:** Επιβάλλει όριο 30 χαρακτήρων για αναγνωριστικά, εμφανίζοντας σφάλμα αν το όριο παραβιαστεί.

## 2.5. Επιπλέον Λεπτομέρειες

**Υποστήριξη Ελληνικών Χαρακτήρων**: Ο Lexer υποστηρίζει πλήρως ελληνικούς χαρακτήρες, τόσο σε αναγνωριστικά (**identifiers**) όσο και σε δεσμευμένες λέξεις

(**keywords**). Αυτό επιτυγχάνεται μέσω της χρήσης **Unicode** και της σωστής διαχείρισης των χαρακτήρων κατά την ανάλυση.

## 2.6. Παράδειγμα Χρήσης

Παρακάτω φαίνεται ένα παράδειγμα εκτέλεσης του λεκτικού αναλυτή για το πρόγραμμα.gr αρχείο που φαίνεται και αυτό παρακάτω:

```
[DEBUG] Tokens:
Token(TokenType.KEYWORD, πρόγραμμα, 1, 1)
Token(TokenType.IDENTIFIER, Παράδειγμα, 1, 2)
Token(TokenType.KEYWORD, δήλωση, 2, 1)
Token(TokenType.IDENTIFIER, x, 2, 2)
Token(TokenType.SEPARATOR, ,, 2, 2)
Token(TokenType.IDENTIFIER, y, 2, 3)
Token(TokenType.KEYWORD, αρχή_προγράμματος, 3, 1)
Token(TokenType.IDENTIFIER, x, 4, 5)
Token(TokenType.OPERATOR, :=, 4, 6)
Token(TokenType.INTEGER, 10, 4, 7)
Token(TokenType.SEPARATOR, ;, 4, 7)
Token(TokenType.IDENTIFIER, y, 5, 5)
Token(TokenType.OPERATOR, :=, 5, 6)
Token(TokenType.IDENTIFIER, x, 5, 7)
Token(TokenType.OPERATOR, +, 5, 8)
Token(TokenType.INTEGER, 5, 5, 9)
Token(TokenType.SEPARATOR, ;, 5, 9)
Token(TokenType.KEYWORD, γράψε, 6, 5)
Token(TokenType.IDENTIFIER, y, 6, 6)
Token(TokenType.KEYWORD, τέλος_προγράμματος, 7, 1)
Token(TokenType.EOF, None, 8, 1)
```

```
πρόγραμμα Παράδειγμα
δήλωση x, y
αρχή_προγράμματος
    x := 10;
    y := x + 5;
    γράψε y
τέλος_προγράμματος
```

# 3. Συντακτικός Αναλυτής

## 3.1. Εισαγωγή

Η κλάση **Parser** αποτελεί ένα βασικό μέρος ενός μεταγλωττιστή, με κύριο ρόλο την ανάλυση της συντακτικής δομής του πηγαίου κώδικα που γράφεται σε μια γλώσσα. Η κλάση **Parser** στον μεταγλωττιστή μας είναι υπεύθυνη για την ανάλυση της σύνταξης (**syntax analysis**) του προγράμματος που γράφτηκε στη γλώσσα μας. Ουσιαστικά, λαμβάνει ως είσοδο μια λίστα από **tokens** που έχουν παραχθεί από

τον **Lexer** και ελέγχει αν η ακολουθία τους ακολουθεί τους γραμματικούς κανόνες της γλώσσας. Παράλληλα, κατασκευάζει τις απαραίτητες δομές για τη σημασιολογική ανάλυση, όπως ο πίνακας συμβόλων (**symbol table**) και ο ενδιάμεσος κώδικας (**intermediate code**).

## 3.2. Γενική Περιγραφή της Κλάσης

Η κλάση **Parser** αρχικοποιείται με μια λίστα από **tokens** και κάποια άλλα μεταδεδομένα όπως το όνομα του αρχείου, το κείμενο του πηγαίου κώδικα και επιλογές για την εμφάνιση προειδοποιήσεων και πληροφοριών. Βασικά χαρακτηριστικά της κλάσης περιλαμβάνουν:

- Έλεγχος της σωστής δήλωσης και χρήσης μεταβλητών, συναρτήσεων και διαδικασιών.
- Εντοπισμός συντακτικών λαθών και εμφάνιση κατάλληλων μηνυμάτων σφάλματος με πληροφορίες για τη θέση τους στον κώδικα (γραμμή και στήλη).
- Ενημέρωση του πίνακα συμβόλων με κάθε νέα δήλωση και διαχείριση των scopes (εμβέλεια).
- Δημιουργία ενδιάμεσου κώδικα (**τετράδες/quads**) που θα χρησιμοποιηθεί σε επόμενα στάδια μεταγλώττισης.
- Πραγματοποίηση ελέγχου τύπων, αρχικοποίησης μεταβλητών και ορθότητας κλήσεων υποπρογραμμάτων και συναρτήσεων.

Με αυτόν τον τρόπο, ο **Parser** αποτελεί το βασικό συνδετικό κρίκο ανάμεσα στη λεξική ανάλυση και στη δημιουργία του ενδιάμεσου κώδικα, διασφαλίζοντας ότι το πρόγραμμα είναι συντακτικά και σημασιολογικά ορθό πριν προχωρήσει στη φάση της παραγωγής τελικού κώδικα.

## 3.3. Λειτουργικότητα και Μέθοδοι

### 3.3.1. Αρχικοποίηση και Βασικές Λειτουργίες

Ο **constructor (__init__)** της κλάσης φορτώνει τα tokens και αρχικοποιεί τις απαραίτητες δομές δεδομένων. Οι βασικές μέθοδοι που χρησιμοποιούνται για την επεξεργασία λαθών και την εμφάνιση μηνυμάτων είναι:

- **error(message):** Εμφανίζει ένα σφάλμα με λεπτομέρειες (**γραμμή, στήλη**).
- **warning(message):** Εμφανίζει μια προειδοποίηση, χωρίς να διακόπτει την εκτέλεση, και έχει χρησιμοποιηθεί στην σημασιολογική ανάλυση.
- **info(message):** Εμφανίζει πληροφοριακά μηνύματα για *debugging* και εμφανίζονται **μόνο** αν το επιλέξει ο χρήστης.

### 3.3.2. Σημασιολογικοί Έλεγχοι

Οι έλεγχοι αυτοί διασφαλίζουν την ορθή σημασιολογία του κώδικα:

- **check_variable_scope(name):** Επιβεβαιώνει ότι μια μεταβλητή είναι προσβάσιμη στην τρέχουσα εμβέλεια (scope), αλλιώς εκτυπώνει μήνυμα σφάλματος.
- **check_declared(name):** Ελέγχει αν ένα αναγνωριστικό (identifier) έχει δηλωθεί πριν τη χρήση του, και αν όχι εκτυπώνει το κατάλληλο μήνυμα σφάλματος.
- **check_assignment(target, value_type):** Επιβεβαιώνει ότι η ανάθεση είναι συμβατή με τους τύπους των μεταβλητών, αλλιώς εκτυπώνει μήνυμα σφάλματος (στην περίπτωσή μας ότι η ανάθεση σε μεταβλητή είναι όντως τύπου Integer).
- **check_operation(left_type, right_type, op):** Επικυρώνει ότι οι πράξεις γίνονται μεταξύ συμβατών τύπων. Με λίγα λόγια ότι το αριστερό μέρος της πράξης και το δεξιό συμβαδίζουν, αλλιώς εκτυπώνει μήνυμα σφάλματος (στην περίπτωσή μας Integer).

### 3.3.3 Διαχείριση Μεταβλητών και Συμβόλων

- **add_variable(name, is_param, par_mode):** Αυτή η μέθοδος προσθέτει μια νέα μεταβλητή στον πίνακα συμβόλων (**symbol table**) του τρέχοντος **scope:**
    - Αν η μεταβλητή είναι παράμετρος **υποπρογράμματος** (**is_param=True**), καταγράφεται ως τέτοια και το **par_mode** καθορίζει αν είναι παράμετρος εισόδου ("**cv**" για call by value) ή εξόδου ("**ref**" για call by reference).
    - Η μέθοδος ελέγχει για **shadowing** και διπλές δηλώσεις, ενημερώνει το offset της μεταβλητής στη μνήμη, και αποθηκεύει πληροφορίες όπως το όνομα, τον τύπο, το **scope** και τη θέση δήλωσης (γραμμή/στήλη).
    - Επίσης, αν πρόκειται για παράμετρο, ενημερώνει τη σχετική λίστα **modes** ώστε να γνωρίζει το **υποπρόγραμμα** τον τρόπο περάσματος κάθε παραμέτρου.
- **add_temp(temp_name):** Η μέθοδος αυτή δημιουργεί μια προσωρινή (**temporary**) μεταβλητή, η οποία χρησιμοποιείται για την αποθήκευση ενδιάμεσων αποτελεσμάτων κατά την παραγωγή του ενδιάμεσου κώδικα (π.χ. **αποτελέσματα πράξεων σε εκφράσεις**):

- Οι προσωρινές μεταβλητές έχουν ειδικό τύπο (**SymbolType.TEMPORARY**) και το όνομά τους είναι μοναδικό (**π.χ. T_1, T_2 κ.λπ**.).
- Καταχωρούνται στον πίνακα συμβόλων με το κατάλληλο offset, ώστε να μπορούν να χρησιμοποιηθούν σωστά κατά τη μεταγλώττιση και στη φάση παραγωγής τελικού κώδικα.

### 3.3.4 Ανάλυση Δομών της Γλώσσας

Η κλάση περιλαμβάνει μεθόδους για την ανάλυση των δομών της γλώσσας, όπως:

- **parse_program():** Η μέθοδος αυτή είναι το σημείο εκκίνησης της ανάλυσης. Ελέγχει αν το πρώτο **token** του αρχείου είναι η λέξη-κλειδί "***πρόγραμμα***". Αν δεν τη βρει, σταματάει αμέσως την εκτέλεση και εμφανίζει συντακτικό σφάλμα, ώστε να διασφαλιστεί ότι το πρόγραμμα ξεκινά με τη σωστή δομή. Αν βρει τη λέξη-κλειδί, διαβάζει το όνομα του προγράμματος και προχωρά στην ανάλυση του κύριου μπλοκ.
- **parse_programblock():** Αυτή η μέθοδος αναλύει το κύριο μέρος του προγράμματος. Ξεκινά με την ανάλυση των δηλώσεων μεταβλητών και των **υποπρογραμμάτων** (συναρτήσεων και διαδικασιών). Στη συνέχεια, ελέγχει αν υπάρχει η λέξη-κλειδί "***αρχή_προγράμματος***" που σηματοδοτεί την αρχή του εκτελέσιμου μέρους. Αφού αναλυθεί η ακολουθία εντολών, ελέγχει για την παρουσία της λέξης-κλειδί "***τέλος_προγράμματος***", που δηλώνει το τέλος του κύριου μπλοκ. Αν λείπει κάποια από αυτές τις λέξεις-κλειδιά, εμφανίζεται συντακτικό σφάλμα.
- **parse_declarations():** Η μέθοδος αυτή επεξεργάζεται όλες τις δηλώσεις μεταβλητών που βρίσκονται στην αρχή του μπλοκ. Ελέγχει διαδοχικά για τη λέξη-κλειδί "***δήλωση***" και για κάθε τέτοια δήλωση, και αναλύει τα ονόματα των μεταβλητών που ακολουθούν. Ενημερώνει τον πίνακα συμβόλων με τις νέες μεταβλητές, ελέγχει για διπλές δηλώσεις ή shadowing, και διασφαλίζει ότι όλες οι μεταβλητές δηλώνονται σωστά πριν χρησιμοποιηθούν.
- **parse_subprograms():** Αυτή η μέθοδος αναλύει τα υποπρογράμματα, δηλαδή τις συναρτήσεις ("***συνάρτηση***") και τις διαδικασίες ("***διαδικασία***"). Για κάθε τέτοια λέξη-κλειδί που συναντά, καλεί τις αντίστοιχες μεθόδους ανάλυσης (**parse_func** για συναρτήσεις, **parse_proc** για διαδικασίες). Ελέγχει τη σωστή δήλωση, τα ορίσματα, το **scope** και το εσωτερικό κάθε **υποπρογράμματος**, και ενημερώνει τον πίνακα συμβόλων με τις σχετικές πληροφορίες.
- **parse_varlist():** Η μέθοδος αναλύει μια λίστα μεταβλητών κατά τη δήλωση ή ορισμό παραμέτρων σε μια συνάρτηση ή διαδικασία. Ελέγχει αν το τρέχον

token είναι αναγνωριστικό (**δηλαδή όνομα μεταβλητής**) και, αν ναι, το προσθέτει στο σύνολο των δηλωμένων μεταβλητών και στον πίνακα συμβόλων (symbol table), λαμβάνοντας υπόψη αν πρόκειται για παράμετρο και τον τρόπο περάσματος (**κατά τιμή ή κατά αναφορά**). Εάν η μεταβλητή είναι παράμετρος, ενημερώνει και τη στοίβα με τους τρόπους παραμέτρων **(param_modes_stack)**. Στη συνέχεια, διαβάζει διαδοχικά επιπλέον μεταβλητές που χωρίζονται με κόμμα, επαναλαμβάνοντας τη διαδικασία για κάθε μία. Αν δεν βρει αναγνωριστικό όπου αναμένεται, εμφανίζει μήνυμα σφάλματος. Έτσι, η μέθοδος αυτή διασφαλίζει ότι όλες οι μεταβλητές ή παράμετροι μιας δήλωσης αναγνωρίζονται, καταχωρούνται σωστά και ελέγχονται για συντακτικά λάθη.

- **parse_func():** Η μέθοδος αναλαμβάνει την ανάλυση και επεξεργασία της δήλωσης μιας συνάρτησης στη γλώσσα. Αρχικά, καταναλώνει το token της λέξης-κλειδιού και ελέγχει αν ακολουθεί ένα αναγνωριστικό, το οποίο θεωρεί ως όνομα της συνάρτησης. Δημιουργεί μια νέα εγγραφή για τη συνάρτηση στον πίνακα συμβόλων, με προκαθορισμένο τύπο επιστροφής (**ακέραιος**). Προσθέτει μια νέα κενή λίστα στη στοίβα των τρόπων παραμέτρων (**param_modes_stack**) και εισέρχεται σε νέο πεδίο ορατότητας (**scope**) για το σώμα της συνάρτησης. Αν ακολουθεί παρένθεση, αναλύει τη λίστα των παραμέτρων. Στη συνέχεια, καλεί τη μέθοδο που αναλύει το σώμα της συνάρτησης. Τέλος, ενημερώνει την εγγραφή της συνάρτησης με το μήκος του πλαισίου (**framelength**) και τους τρόπους παραμέτρων, παράγει το κατάλληλο ενδιάμεσο κώδικα για το τέλος του block και εξέρχεται από το πεδίο ορατότητας.

- **parse_proc():** Η μέθοδος αφορά τη δήλωση διαδικασίας (**procedure**), δηλαδή **υποπρογράμματος** που δεν επιστρέφει τιμή. Καταναλώνει το token της λέξης-κλειδιού, ελέγχει για το όνομα της διαδικασίας και δημιουργεί αντίστοιχη εγγραφή στον πίνακα συμβόλων. Προσθέτει μια νέα λίστα στη στοίβα των τρόπων παραμέτρων και εισέρχεται σε νέο scope. Αν υπάρχει λίστα παραμέτρων, την αναλύει. Έπειτα, καλεί τη μέθοδο που αναλύει το σώμα της διαδικασίας. Τέλος, ενημερώνει την εγγραφή της διαδικασίας με το framelength και τους τρόπους παραμέτρων (cv ή ref), παράγει ενδιάμεσο κώδικα για το τέλος του block και εξέρχεται από το scope.

- **parse_if_stat():** Η μέθοδος αναλύει μια εντολή επιλογής (**if statement**) στη γλώσσα. Αρχικά, καταναλώνει το token της λέξης-κλειδιού "**εάν**" και αναλύει τη συνθήκη με τη **parse_condition**. Αν ακολουθεί η λέξη-κλειδί "τότε", προχωρά στην ανάλυση της ακολουθίας εντολών του "then" μπλοκ. Χρησιμοποιεί τεχνικές backpatching για να συνδέσει τα σωστά σημεία

εκτέλεσης ανάλογα με το αποτέλεσμα της συνθήκης. Αν υπάρχει "**αλλιώς**", αναλύει και το else μπλοκ, δημιουργώντας κατάλληλα jump quads ώστε να παρακάμπτεται το else όταν η συνθήκη είναι αληθής. Τέλος, απαιτεί το **εάν_τέλος** για να ολοκληρωθεί σωστά η δομή. Αν λείπει κάποιο από τα απαραίτητα tokens, εμφανίζει συντακτικό σφάλμα.

- **parse_while_stat():** Η μέθοδος αναλύει μια εντολή επανάληψης τύπου while (**όσο ... επανάλαβε ... όσο_τέλος**). Καταναλώνει το token "**όσο**", σημειώνει τη θέση έναρξης της συνθήκης και αναλύει τη συνθήκη με τη parse_condition. Αν ακολουθεί το "επανάλαβε", αναλύει το σώμα του βρόχου. Μετά το σώμα, παράγει quad για άλμα πίσω στην αρχή της συνθήκης και κάνει backpatch τις false λίστες ώστε να οδηγούν στο σημείο εξόδου από τον βρόχο. Τέλος, απαιτεί το "**όσο_τέλος**" για να κλείσει σωστά η δομή. Αν λείπει κάποιο από τα απαραίτητα tokens, εμφανίζει συντακτικό σφάλμα.

- **parse_do_stat():** Η μέθοδος αναλύει μια εντολή επανάληψης τύπου do-while. Αρχικά, καταναλώνει το token της λέξης-κλειδιού (π.χ. "επανάλαβε"), σημειώνει τη θέση του πρώτου quad (για να μπορεί να επιστρέψει εκεί), και αναλύει το σώμα της επανάληψης με parse_sequence. Στη συνέχεια, περιμένει τη λέξη-κλειδί "μέχρι" και αναλύει τη συνθήκη τερματισμού. Με το backpatching, συνδέει τα true της συνθήκης στην αρχή του βρόχου (ώστε αν η συνθήκη ισχύει να επαναληφθεί) και τα false στο επόμενο quad (έξοδος από τον βρόχο). Αν λείπει το "μέχρι", εμφανίζει συντακτικό σφάλμα.

- **parse_for_stat()**: Η μέθοδος αναλύει μια εντολή επανάληψης τύπου for. Καταναλώνει το "για" και περιμένει ένα αναγνωριστικό ως μεταβλητή ελέγχου του βρόχου. Ελέγχει για τον τελεστή ανάθεσης ":=", αναλύει την αρχική τιμή και παράγει το αντίστοιχο quad. Στη συνέχεια, περιμένει τη λέξη-κλειδί "έως" και αναλύει την τελική τιμή. Προαιρετικά, μπορεί να ακολουθεί "**με_βήμα**" για να ορίσει το βήμα της επανάληψης (διαφορετικά το βήμα είναι 1). Δημιουργεί quads για τη σύγκριση της μεταβλητής με το όριο, και για τα άλματα μέσα και έξω από το βρόχο. Αναλύει το σώμα της επανάληψης, παράγει quad για την αύξηση της μεταβλητής κατά το βήμα, και άλμα πίσω στη σύγκριση. Τέλος, απαιτεί το "**για_τέλος**" για να κλείσει σωστά η δομή. Αν λείπει κάποιο απαραίτητο στοιχείο, εμφανίζει συντακτικό σφάλμα.

### 3.3.5. Ανάλυση Εκφράσεων και Προτάσεων

- **parse_expression():** Η μέθοδος αυτή αναλαμβάνει να αναλύσει αριθμητικές και λογικές εκφράσεις. Ξεκινά διαβάζοντας προαιρετικό πρόσημο (**+ ή -**), και στη συνέχεια αναλύει όρους (**terms**) και παράγοντες

(**factors**) που μπορεί να συνδέονται με τελεστές όπως **+, -, *, /**. Κατά την ανάλυση, γίνεται έλεγχος τύπων ώστε να διασφαλιστεί ότι οι πράξεις είναι έγκυρες. Τέλος, παράγεται ο αντίστοιχος ενδιάμεσος κώδικας (**quads**) για κάθε πράξ (θα συζητηθεί σε επόμενη φάση της αναφοράς).

- **parse_statement():** Επεξεργάζεται προτάσεις όπως ανάθεση, κλήσεις συναρτήσεων και δομές ελέγχου:
    - Εκχώρηση τιμής σε μεταβλητή (**assignment**)
    - Κλήση συνάρτησης ή διαδικασίας
    - Εντολές εισόδου/εξόδου (**διάβασε, γράψε**)
    - Δομές ελέγχου ροής όπως **εάν**, **όσο**, **για**, **επανάλαβε ... μέχρι.** Για κάθε τύπο πρότασης, καλεί την αντίστοιχη μέθοδο (π.χ. **parse_if_stat**, **parse_while_stat**, **parse_assignment_stat**), ελέγχει τη συντακτική ορθότητα και στο τέλος παράγει τον κατάλληλο ενδιάμεσο κώδικα.
- **parse_condition():** Η μέθοδος αυτή χειρίζεται τη σύνταξη και τη σημασιολογία των λογικών συνθηκών. Υποστηρίζει σύνθετες συνθήκες με λογικούς τελεστές "***και***", "***ή***", καθώς και άρνηση ("***όχι***"). Αναλύει κάθε όρο της συνθήκης, δημιουργεί τις απαραίτητες λίστες για **true/false** διακλαδώσεις (**backpatching**) και παράγει **quads** για τις λογικές πράξεις και τις συγκρίσεις. Έτσι, εξασφαλίζεται η σωστή εκτέλεση των δομών ελέγχου που βασίζονται σε συνθήκες.
- **parse_assignment_stat:** Η μέθοδος αναλύει και ελέγχει συντακτικά και σημασιολογικά μια εντολή ανάθεσης (**assignment statement**). Αρχικά, ελέγχει αν το τρέχον token είναι αναγνωριστικό (**δηλαδή όνομα μεταβλητής ή συνάρτησης**). Εάν ναι, αναζητά το όνομα στον πίνακα συμβόλων και ελέγχει αν πρόκειται για όνομα συνάρτησης (ώστε να υποστηρίξει την ανάθεση τιμής επιστροφής σε συνάρτηση). Στη συνέχεια, καταναλώνει το αναγνωριστικό και ελέγχει αν ακολουθεί ο τελεστής ανάθεσης **:=**. Εάν υπάρχει, αναλύει τη δεξιά πλευρά της ανάθεσης ως έκφραση, προσδιορίζει τον τύπο της και ελέγχει αν ο τύπος της έκφρασης είναι συμβατός με τον τύπο της μεταβλητής (ή της συνάρτησης). Έτσι, διασφαλίζεται η ορθότητα και η ασφάλεια τύπων στις εντολές ανάθεσης.
- **parse_factor():** Η συνάρτηση parse_factor αναλύει έναν παράγοντα (factor) μιας έκφρασης, πραγματοποιώντας και σημασιολογικούς ελέγχους. Συγκεκριμένα:
    - Αν το τρέχον token είναι ακέραιος (**INTEGER**) επιστρέφει την αριθμητική του τιμή.
    - Αν το token είναι αριστερή παρένθεση **"("**, αναλύει αναδρομικά μια έκφραση και απαιτεί να ακολουθεί δεξιά παρένθεση, ελέγχοντας έτσι την ισορροπία των παρενθέσεων.

- Αν το token είναι αναγνωριστικό (**IDENTIFIER**), ελέγχει αν ακολουθεί παρένθεση, οπότε το θεωρεί ως κλήση συνάρτησης ή διαδικασίας. Σε αυτή την περίπτωση, ελέγχει αν το όνομα είναι δηλωμένο και αν είναι πράγματι συνάρτηση ή διαδικασία.
- Αν το αναγνωριστικό δεν ακολουθείται από παρένθεση, θεωρείται μεταβλητή. Ελέγχει αν είναι δηλωμένη και αν είναι πράγματι μεταβλητή, παράμετρος ή προσωρινή μεταβλητή.
- Σε κάθε άλλη περίπτωση, εμφανίζει μήνυμα σφάλματος.

- **parse_optionalsign():** Η μέθοδος αναλύει ένα προαιρετικό πρόσημο (+ ή - ) που μπορεί να προηγείται από έναν αριθμό ή έκφραση. Αν το τρέχον token είναι ο τελεστής -, καταναλώνει το token και επιστρέφει -1, δηλώνοντας ότι το πρόσημο είναι αρνητικό. Αν είναι +, απλώς το καταναλώνει και επιστρέφει 1 (θετικό πρόσημο). Αν δεν υπάρχει πρόσημο, επιστρέφει επίσης 1, θεωρώντας το πρόσημο θετικό από προεπιλογή.
- **parse_idtail():** Η μέθοδος αναλύει το "υπόλοιπο" ενός αναγνωριστικού, δηλαδή ελέγχει αν μετά από ένα όνομα ακολουθεί παρένθεση **(**, κάτι που δηλώνει κλήση συνάρτησης ή διαδικασίας. Αν βρει παρένθεση, καλεί τη **parse_actualpars** για να αναλύσει τα πραγματικά ορίσματα της κλήσης. Αν δεν υπάρχει παρένθεση, δεν κάνει τίποτα, καθώς το αναγνωριστικό θεωρείται απλή μεταβλητή.
- **parse_boolterm:** Η μέθοδος αναλύει μια λογική έκφραση που συνδέει λογικούς παράγοντες με τον τελεστή "**και**". Ξεκινά αναλύοντας έναν λογικό παράγοντα (**parse_boolfactor**) και, όσο το επόμενο token είναι η λέξη-κλειδί "**και**", συνεχίζει να αναλύει επιπλέον λογικούς παράγοντες. Για κάθε νέο παράγοντα, κάνει backpatch τις λίστες true του αριστερού όρου στην αρχή του δεξιού, συγχωνεύει τις false λίστες και ενημερώνει τη true λίστα με αυτή του δεξιού όρου. Έτσι, υλοποιείται σωστά η βραχυκύκλωση (short-circuit evaluation) του "**και**".
- **parse_boolfactor:** Η μέθοδος αναλύει έναν λογικό παράγοντα. Αν βρει το "**όχι**", περιμένει μια έκφραση μέσα σε αγκύλες, αναλύει την συνθήκη και επιστρέφει ένα αντικείμενο όπου οι λίστες true και false έχουν αντιστραφεί (υλοποιώντας το λογικό **NOT**). Αν βρει απλώς αγκύλες, αναλύει την συνθήκη μέσα σε αυτές και την επιστρέφει. Σε κάθε άλλη περίπτωση, περιμένει μια έκφραση ακολουθούμενη από έναν συγκριτικό τελεστή (**π.χ. =, <, >, <=, >=, <>**), αναλύει και τη δεξιά έκφραση, και παράγει τα κατάλληλα quads για τη σύγκριση και το άλμα. Επιστρέφει ένα αντικείμενο με τις λίστες true και false για χρήση σε σύνθετες λογικές εκφράσεις. Αν δεν βρει συγκριτικό τελεστή, εμφανίζει σφάλμα.
- **parse_input_stat():** Η μέθοδος αναλύει μια εντολή εισόδου (**π.χ. "διάβασε**"). Αρχικά, καταναλώνει το token της λέξης-κλειδιού και ελέγχει αν ακολουθεί αναγνωριστικό, το οποίο θεωρεί ως όνομα μεταβλητής.

Προσθέτει το όνομα στις χρησιμοποιούμενες μεταβλητές, καταναλώνει το αναγνωριστικό και παράγει το αντίστοιχο quad για είσοδο τιμής στη μεταβλητή. Αν δεν βρει αναγνωριστικό, εμφανίζει συντακτικό σφάλμα.

- **parse_print_stat():** Η μέθοδος αναλύει μια εντολή εξόδου (**π.χ. "γράψε**"). Καταναλώνει το token της λέξης-κλειδιού, αναλύει την έκφραση που ακολουθεί και παράγει quad για εκτύπωση της τιμής της έκφρασης.
- **parse_call_stat():** Η μέθοδος αναλύει μια εντολή κλήσης διαδικασίας ή συνάρτησης **(π.χ. "εκτέλεσε").** Καταναλώνει το token της λέξης-κλειδιού, ελέγχει για αναγνωριστικό (**όνομα διαδικασίας/συνάρτησης**), το καταναλώνει, αναλύει τυχόν παραμέτρους και παράγει quad για την κλήση. Αν δεν βρει αναγνωριστικό, εμφανίζει σφάλμα.

### 3.3.6. Διαχείριση Ενδιάμεσου Κώδικα

Η κλάση **Parser** συνεργάζεται στενά με την **IntermediateCodeGenerator** για τη δημιουργία και διαχείριση του ενδιάμεσου κώδικα του προγράμματος. Ο ενδιάμεσος κώδικας αποτελείται από τετράδες (**quads**), οι οποίες περιγράφουν με αφηρημένο τρόπο τις βασικές εντολές του προγράμματος, ανεξάρτητα από την τελική γλώσσα μηχανής ή assembly. Περισσότερες λεπτομέρειες θα δωθούν στην συνέχεια της αναφοράς.

## 3.4. Παράδειγμα Χρήσης

Για παράδειγμα, η ανάλυση μιας απλής συνάρτησης μπορεί να γίνει ως εξής:

1. Η **parse_func()** αναγνωρίζει τη λέξη-κλειδί "***συνάρτηση***" και δημιουργεί μια νέα εγγραφή στον πίνακα συμβόλων.
2. Η **parse_formalparlist()** επεξεργάζεται τις παραμέτρους της συνάρτησης.
3. Η **parse_funcblock()** αναλύει το σώμα της συνάρτησης, συμπεριλαμβανομένων των δηλώσεων και των εντολών της.
4. Η **check_return_paths()** ελέγχει ότι όλες οι πιθανές διαδρομές της συνάρτησης επιστρέφουν μια τιμή, και αν διαπιστώσει πως όχι, τότε εκτυπώνει μήνυμα προειδοποίησης.

## 3.5. Παράδειγμα Λανθασμένης Σύνταξης

```
πρόγραμμα Παράδειγμα
δήλωση x, y
αρχή_προγράμματος
    x = 10;
    y := x + 5;
    γράψε y
τέλος_προγράμματος
```

Στα αριστερά βλέπουμε ένα λανθασμένο παράδειγμα σύνταξης όπου αντί := ο χρήστης τοποθετεί =, πράγμα που σύμφωνα με την γραμματική της greek++ πρέπει να προκαλεί σφάλμα. Παρακάτω βλέπουμε ένα παράδειγμα από το παραγόμενο σφάλμα. Στο σφάλμα αυτό αναγράφεται και η γραμμή αλλά, η στήλη και

αναλυτικό μήνυμα του συντακτικού λάθους, ώστε να καταλαβαίνει ο χρήστης ευκολότερα για τι λάθος πρόκειται:

```
[ERROR] παράδειγμα.gr:4:6: Expected ':=' in assignment statement
        x = 10;
          ^
```

```
πρόγραμμα
δήλωση x, y
αρχή_προγράμματος
    x := 10;
    y := x + 5;
    γράψε y
τέλος_προγράμματος
```

Παρομοίως για αυτό το πρόγραμμα όπου και λείπει ο **identifier** δίπλα από το **keyword** πρόγραμμα. Ο χρήστης έπρεπε να είχε βάλει **πρόγραμμα Παράδειγμα.** Παρόλο που το error δείχνει στην δεύτερη γραμμή του προγράμματος, το μήνυμα λάθους κατευθύνει επαρκώς τον χρήστη για την σύνταξη.

```
[ERROR] παράδειγμα.gr:2:1: Expected program name (IDENTIFIER)
    δήλωση x, y
    ^
```

### 3.6. Συμπέρασμα

Συνοψίζοντας, η κλάση **Parser** αποτελεί ένα από τα πιο κρίσιμα και πολύπλοκα τμήματα του μεταγλωττιστή, καθώς ενώνει το στάδιο της λεξικής ανάλυσης με τη δημιουργία του ενδιάμεσου κώδικα και τη σημασιολογική επαλήθευση του προγράμματος. Μέσα από τις μεθόδους της, διασφαλίζει ότι το πρόγραμμα ακολουθεί πιστά τους συντακτικούς και σημασιολογικούς κανόνες της γλώσσας, εντοπίζοντας και αναφέροντας σφάλματα με σαφήνεια και ακρίβεια.

## 4. Σημασιολογική Ανάλυση

### 4.1. Εισαγωγή

Η σημασιολογική ανάλυση αποτελεί **ένα κρίσιμο στάδιο της διαδικασίας μεταγλώττισης**, καθώς ελέγχει την ορθότητα του προγράμματος ως προς τη σημασία του. Σε αυτή τη φάση, επιβεβαιώνουμε ότι οι μεταβλητές έχουν δηλωθεί πριν τη χρήση τους, ότι οι πράξεις είναι συμβατές με τους τύπους των δεδομένων και ότι οι κανόνες της γλώσσας τηρούνται. Παρακάτω περιγράφουμε πώς υλοποιήσαμε τη σημασιολογική ανάλυση στον μεταγλωττιστή μας.

### 4.2. Δομή και Λειτουργία της Σημασιολογικής Ανάλυσης

Η σημασιολογική ανάλυση βασίζεται στον έλεγχο τύπων, στη διαχείριση εμβέλειας μεταβλητών και στον έλεγχο σημασιολογικών κανόνων. Οι κύριοι τομείς εστίασης ήταν:

- **Έλεγχος δηλώσεων μεταβλητών**: Επιβεβαιώνεται ότι οι μεταβλητές έχουν δηλωθεί πριν τη χρήση τους και ότι δεν υπάρχουν διπλές δηλώσεις στην ίδια εμβέλεια.
- **Έλεγχος τύπων**: Ελέγχεται η συμβατότητα τύπων σε εκφράσεις, εκχωρήσεις και παραμέτρους συναρτήσεων.
- **Διαχείριση εμβέλειας**: Πραγματοποιείται μέσω ενός πίνακα συμβόλων (**Symbol Table**) που καταγράφει τις μεταβλητές, τις συναρτήσεις και τις παραμέτρους, μαζί με την εμβέλειά τους.

## 4.3. Πίνακας Συμβόλων (Symbol Table)

Ο πίνακας συμβόλων αποτελεί τον πυρήνα της σημασιολογικής ανάλυσης. Περιλαμβάνει τις ακόλουθες λειτουργίες:

- **Εγγραφή συμβόλων**: Κάθε μεταβλητή, συνάρτηση ή παράμετρος καταχωρείται στον πίνακα με πληροφορίες όπως τύπος, εμβέλεια και offset στη μνήμη.
- **Αναζήτηση συμβόλων**: Γίνεται αναζήτηση για επιβεβαίωση ότι ένα σύμβολο έχει δηλωθεί και είναι προσβάσιμο στην τρέχουσα εμβέλεια.
- **Ελέγχος σκίασης (Shadowing)**: Εκδίδεται προειδοποίηση όταν μια μεταβλητή σκιάζει άλλη από εξωτερική εμβέλεια.

## 4.4. Έλεγχος Τύπων (Type Checking)

Ο έλεγχος τύπων πραγματοποιείται σε όλες τις εκφράσεις και τις εκχωρήσεις. Βασικές λειτουργίες:

- **Συμβατότητα τύπων**: Ελέγχεται αν οι τύποι των τελεστών σε πράξεις (**π.χ., +, ***) είναι συμβατοί.
- **Εκχωρήσεις**: Επιβεβαιώνεται ότι η τιμή που εκχωρείται σε μια μεταβλητή είναι συμβατή με τον τύπο της.
- **Συναρτήσεις**: Ελέγχεται αν οι παράμετροι και οι τιμές επιστροφής έχουν τους σωστούς τύπους.

## 4.5. Εκτέλεση Σημασιολογικών Ελέγχων

Κατά τη διάρκεια της ανάλυσης, πραγματοποιούνται οι ακόλουθοι έλεγχοι:

- **Αρχικοποίηση μεταβλητών**: Εκδίδεται προειδοποίηση αν μια μεταβλητή χρησιμοποιείται πριν την αρχικοποίησή της.
- **Συναρτήσεις**: Ελέγχεται αν όλες οι διαδρομές μιας συνάρτησης επιστρέφουν τιμή.
- **Παράμετροι**: Επιβεβαιώνεται ότι οι παράμετροι συναρτήσεων περνιούνται σωστά (**by value ή by reference**).

## 4.6. Παράδειγμα Λειτουργίας

Παρακάτω παραθέτουμε ένα παράδειγμα της λειτουργίας του μέρους της σημασιολογικής ανάλυσης όπου η συνάρτηση που καλείται δεν υπάρχει στο πρόγραμμα:

```
αρχή_προγράμματος
α := 1;
β := 2 + α * α / (2 - α - (2 * α)); {σχόλιο}
γ := αύξηση_αδήλωτη(α, %β);
```

```
ValueError: Undeclared function 'αύξηση_αδήλωτη' at line 24, column 7
```

Παρακάτω παραθέτουμε ένα παράδειγμα της λειτουργίας του μέρους της σημασιολογικής ανάλυσης όπου γίνεται το φαινόμενο **variable shadowing**[1]. Ο μεταγλωττιστής μας επιτρέπει το φαινόμενο αυτό, απλά εμφανίζει **warning** για να προειδοποιήσει τον χρήστη για να αποτρέψει τυχόν μπερδέματα με τις μεταβλητές (**style-checking**)[2].

```
[WARNING] Variable 'α' at line 7, column 11 shadows declaration at line 2
[WARNING] Variable 'β' at line 8, column 11 shadows declaration at line 2
```

```
πρόγραμμα τεστ
    δήλωση α, β
    δήλωση γ

    συνάρτηση αύξηση(α, β)
        διαπροσωπεία
                είσοδος α
                έξοδος β
        αρχή_συνάρτησης
            β := α + 1;
            αύξηση := α + 1
        τέλος_συνάρτησης
```

## 4.7. Συμπέρασμα

Η σημασιολογική ανάλυση υλοποιήθηκε με επιτυχία μέσω ενός συνδυασμού πίνακα συμβόλων, ελέγχων τύπων και διαχείρισης εμβέλειας. Οι έλεγχοι που πραγματοποιούνται εξασφαλίζουν ότι το πρόγραμμα είναι σημασιολογικά ορθό και πληροί τους κανόνες της γλώσσας. Παράλληλα, οι προειδοποιήσεις για πιθανά προβλήματα (**όπως η σκίαση μεταβλητών**) βοηθούν τον προγραμματιστή να γράψει καθαρότερο κώδικα.

---

[1] Variable shadowing - Wikipedia
[2] Checkstyle - Wikipedia

# 5. Παραγωγή Ενδιάμεσου Κώδικα

## 5.1. Εισαγωγή

Η κλάση **IntermediateCodeGenerator** αποτελεί ένα βασικό στοιχείο του μεταγλωττιστή μας, καθώς είναι υπεύθυνη για τη δημιουργία του ενδιάμεσου κώδικα. Ο ενδιάμεσος κώδικας αναπαρίσταται ως μια ακολουθία τετράδων (**quads**), όπου κάθε τετράδα περιλαμβάνει έναν τελεστή και έως τρία ορίσματα. Αυτή η αναπαράσταση είναι αφηρημένη και ανεξάρτητη από την αρχιτεκτονική της τελικής μηχανής, γεγονός που την καθιστά ιδανική για περαιτέρω βελτιστοποιήσεις και μετατροπή σε κώδικα μηχανής.

## 5.2. Γενική Περιγραφή της Κλάσης

Η κλάση παρέχει μεθόδους για τη δημιουργία, διαχείριση και βελτιστοποίηση αυτών των τετράδων, καθώς και για τη διαχείριση προσωρινών μεταβλητών, ετικετών (**labels**) και λιστών για **backpatching**. Επιπλέον, προσφέρει δυνατότητες για την αποθήκευση και εκτύπωση του ενδιάμεσου κώδικα, διευκολύνοντας τον έλεγχο και την ανάλυση του παραγόμενου αποτελέσματος. Μέσω της συνεργασίας της με τον **parser** και τον πίνακα συμβόλων, η **IntermediateCodeGenerator** διασφαλίζει ότι το πρόγραμμα μετατρέπεται σε μια συνεπή και λειτουργική ενδιάμεση μορφή, έτοιμη για τα επόμενα στάδια της μεταγλώττισης.

## 5.3. Λειτουργικότητα και Μέθοδοι

Η κλάση προσφέρει μια σειρά από μεθόδους για τη δημιουργία και διαχείριση του ενδιάμεσου κώδικα:

### 5.3.1. Δημιουργία και Διαχείριση Τετράδων

- **genquad(op, x, y, z):** Δημιουργεί μια νέα τετράδα (**quad**) με τον τελεστή **op** και τα ορίσματα **x, y, z**. Η τετράδα αυτή προστίθεται στη λίστα του ενδιάμεσου κώδικα και επιστρέφει το index της. Είναι η βασική μέθοδος για την καταγραφή κάθε εντολής του προγράμματος σε ενδιάμεση μορφή.
- **nextquad():** Επιστρέφει το **index της επόμενης τετράδας** που θα παραχθεί. Χρησιμοποιείται για τον έλεγχο της ροής του προγράμματος και για τη σωστή διαχείριση διακλαδώσεων και **backpatching**.

### 5.3.2. Διαχείριση Προσωρινών Μεταβλητών

- **newtemp():** Παράγει ένα νέο όνομα προσωρινής μεταβλητής (**π.χ. T_1, T_2**) και το καταχωρεί στον πίνακα συμβόλων μέσω του **parser**. Οι προσωρινές μεταβλητές χρησιμοποιούνται για την αποθήκευση

ενδιάμεσων αποτελεσμάτων σε εκφράσεις και πράξεις, διευκολύνοντας τη διαχείριση σύνθετων υπολογισμών.

### 5.3.3. Διαχείριση Διακλαδώσεων και Backpatching

- **backpatch(quad_list, target):** Ενημερώνει τις τετράδες που βρίσκονται στη λίστα **quad_list**, θέτοντας ως στόχο (**jump target**) το **target**. Είναι απαραίτητη για τη σωστή υλοποίηση των δομών ελέγχου ροής, όπου οι διευθύνσεις διακλάδωσης δεν είναι γνωστές κατά τη δημιουργία των **quads**.
- **makelist(x):** Δημιουργεί μια νέα λίστα που περιέχει το **index** μιας τετράδας. Χρησιμοποιείται για τη διαχείριση των **true/false lists** κατά το **backpatching**, διευκολύνοντας τη σύνδεση των διακλαδώσεων.
- **merge(list1, list2):** Συγχωνεύει δύο λίστες από **indices** τετράδων, διευκολύνοντας τη διαχείριση πολλαπλών διακλαδώσεων και την οργάνωση των **true/false lists**.

### 5.3.4. Εμφάνιση και Αποθήκευση Ενδιάμεσου Κώδικα

- **print_intermediate_code():** Εκτυπώνει τον ενδιάμεσο κώδικα σε αναγνώσιμη μορφή, ώστε να μπορεί να ελεγχθεί και παρέχει μια σαφή εικόνα της ενδιάμεσης αναπαράστασης του προγράμματος.
- **save_intermediate_code(output_file):** Αποθηκεύει τον ενδιάμεσο κώδικα σε αρχείο, για περαιτέρω ανάλυση ή χρήση από τα επόμενα στάδια της μεταγλώττισης. Είναι χρήσιμη για **debugging** και για την τελική παραγωγή κώδικα.

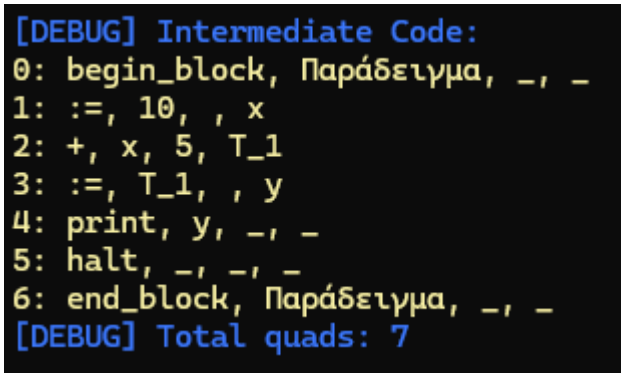
## 5.4. Παράδειγμα Χρήσης

Παρακάτω επισυνάπτουμε ένα παράδειγμα χρήσης του Ενδιάμεσου Κώδικα για το αρχείο **παράδειγμα.gr.** Η δεξιά φωτογραφία αναπαριστά το αρχείο **παράδειγμα.gr**, ενώ η αριστερά το **παράδειγμα.int**:

```
0: begin_block, Παράδειγμα, _, _
1: :=, 10, , x
2: +, x, 5, T_1
3: :=, T_1, , y
4: print, y, _, _
5: halt, _, _, _
6: end_block, Παράδειγμα, _, _
```

```
πρόγραμμα Παράδειγμα
δήλωση x, y
αρχή_προγράμματος
    x := 10;
    y := x + 5;
    γράψε y
τέλος_προγράμματος
```

Αν ο προγραμματιστής χρειάζεται να δει τις πληροφορίες για τον ενδιάμεσο κώδικα μέσω του τερματικού, μπορεί να δώσει την εντολή **python3 greek_5387_5388.py παράδειγμα.gr --debug** και θα εμφανιστεί όπως φαίνεται παρακάτω:




Όπως μπορούμε να δούμε στην φωτογραφία στα αριστερά, εμφανίζονται κάποια έξτρα στοιχεία όπως ο συνολικός αριθμός των quads, καθώς και μια χρωματική αναπαράσταση.

# 6. Βελτιστοποίηση Ενδιάμεσου Κώδικα

## 6.1. Ανάλυση της fold_constants()

Η συνάρτηση **fold_constants()** υλοποιεί τη βελτιστοποίηση **constant folding και constant propagation**[3], η οποία:

1. Εντοπίζει πράξεις με σταθερά ορίσματα
2. Υπολογίζει το αποτέλεσμα κατά τη μεταγλώττιση
3. Αντικαθιστά την αρχική τετράδα με απλή ανάθεση της σταθεράς

## 6.2. Πώς λειτουργεί

1. Δημιουργεί έναν χάρτη (**const_map**) με όλες τις γνωστές σταθερές από τον πίνακα συμβόλων.
2. Εξετάζει κάθε τετράδα για πράξεις **(+, -, *, /)** με σταθερά ορίσματα.
3. Για τετράδες ανάθεσης **(:=)**, ενημερώνει τον πίνακα συμβόλων με νέες σταθερές.
4. Αντικαθιστά τις τετράδες με απλοποιημένες εκδοχές τους.

## 6.3. Προβλήματα που αντιμετωπίστηκαν

- **Ασυμβατότητα με την κλάση RIscvCodeGenerator**: Η υλοποίηση δημιουργούσε προβλήματα στον τελικό κώδικα λόγω διαφορών στη διαχείριση των **registers**.
- **Προσωρινές μεταβλητές**: Η σταθερή προπαγάνδευση σε προσωρινές μεταβλητές οδηγούσε σε μη αναμενόμενα αποτελέσματα.
- **Εμβέλεια συμβόλων**: Η σωστή παρακολούθηση της εμβέλειας των σταθερών απαιτούσε περαιτέρω ανάπτυξη.

[3] Constant folding - Wikipedia

## 6.4. Σχόλια

Παρόλο που η συνάρτηση μείωνε σημαντικά τον αριθμό των τετράδων στον ενδιάμεσο κώδικα (**κάποιες φορές έως και 30%**), αποφασίσαμε να την απενεργοποιήσουμε λόγω των προβλημάτων που προκαλούσε στον τελικό κώδικα. Ο κώδικας παραμένει στο αρχείο σε σχόλια ως βάση για μελλοντική βελτίωση.

## 6.5. Παράδειγμα Εκτέλεσης

Παρακάτω παραθέτουμε δύο παραδείγματα εκτέλεσης παραγωγής του ενδιάμεσου κώδικα με **constant folding, constant propagation** (**δεξιά**) και χωρίς αυτές τις τεχνικές (**αριστερά**).

```
0: begin_block, αύξηση, _, _
1: +, α, 1, T_1
2: :=, T_1, , β
3: +, α, 1, T_2
4: :=, T_2, , αύξηση
5: retv, αύξηση, , _
6: end_block, αύξηση, _, _
7: begin_block, τύπωσε_συν_1, _, _
8: +, χ, 1, T_3
9: print, T_3, _, _
10: end_block, τύπωσε_συν_1, _, _
11: begin_block, τεστ, _, _
12: :=, 1, , α
13: *, α, α, T_4
14: -, 2, α, T_5
15: *, 2, α, T_6
16: -, T_5, T_6, T_7
17: /, T_4, T_7, T_8
18: +, 2, T_8, T_9
19: :=, T_9, , β
20: par, α, CV, _
21: par, β, REF, _
22: call, αύξηση, _, T_10
23: :=, T_10, , γ
24: :=, 1, _, α
25: <=, α, 8, 27
26: jump, _, _, 31
27: par, α, CV, _
28: call, τύπωσε_συν_1, _, _
29: +, α, 2, α
30: jump, _, _, 25
31: :=, 1, , β
32: <, β, 10, 34
33: jump, _, _, 43
34: <>, β, 22, 40
35: jump, _, _, 36
36: >=, β, 23, 38
37: jump, _, _, 42
38: <=, β, 24, 40
39: jump, _, _, 42
40: +, β, 1, T_11
41: :=, T_11, , β
42: jump, _, _, 32
43: input, _, _, β
44: +, β, 1, T_12
45: :=, T_12, , β
46: <, β, -100, 44
47: jump, _, _, 48
48: halt, _, _, _
49: end_block, τεστ, _, _
```

```
0: begin_block, αύξηση, _, _
1: +, α, 1, T_1
2: :=, T_1, , β
3: +, α, 1, T_2
4: :=, T_2, , αύξηση
5: retv, αύξηση, , _
6: end_block, αύξηση, _, _
7: begin_block, τύπωσε_συν_1, _, _
8: +, χ, 1, T_3
9: print, T_3, _, _
10: end_block, τύπωσε_συν_1, _, _
11: begin_block, τεστ, _, _
12: :=, 1, , α
13: :=, 1, , β
14: par, α, CV, _
15: par, β, REF, _
16: call, αύξηση, _, T_4
17: :=, T_4, , γ
18: :=, 1, _, α
19: <=, α, 8, 21
20: jump, _, _, 25
21: par, α, CV, _
22: call, τύπωσε_συν_1, _, _
23: +, α, 2, α
24: jump, _, _, 19
25: :=, 1, , β
26: <, β, 10, 28
27: jump, _, _, 36
28: <>, β, 22, 34
29: jump, _, _, 30
30: >=, β, 23, 32
31: jump, _, _, 35
32: <=, β, 24, 34
33: jump, _, _, 35
34: :=, 2, , β
35: jump, _, _, 26
36: input, _, _, β
37: :=, 3, , β
38: <, β, -100, 37
39: jump, _, _, 40
40: halt, _, _, _
41: end_block, τεστ, _, _
```

# 7. Πίνακας Συμβόλων

## 7.1. Εισαγωγή

Η κλάση **SymbolTable** έχει την ευθύνη καταγραφής των μεταβλητών, παραμέτρων, συναρτήσεων και διαδικασιών του προγράμματος το οποίο μεταγλωτίζεται, το οποίο αποτελεί απαραίτητο κομμάτι και για την παραγωγή τελικού κώδικα στο τέλος της μετάφρασης.

## 7.2. Γενική Περιγραφή της Κλάσης

Η κλάση **SymbolTable** αρχικά λαμβάνει ως είσοδο το όνομα του αρχείου το οποίο θα επεξεργάζεται κατά τη λειτουργία της.

Η μεταβλητές που κρατάει είναι:

- **scopes**: ένας πίνακας
- **current_scope_level**: το τρέχων επίπεδο που βρίσκεται το πρόγραμμα όταν του γίνεται η μετάφραση.
- **offset_counter**: το offset το οποίο θα έχουν οι μεταβλητές μέσα στη μνήμη. Ξεκινάει πάντα από το 12 και ανεβαίνει κατά 4 με κάθε νέα μεταβλητή.
- **offset_stack**: ένα stack το οποίο κρατάει προσωρινά τα offset της τελευταίας μεταβλητής ενός επιπέδου. Κάθε φορά που προκύπτει ένα νέο επίπεδο, προστίθεται το offset του τωρινού επιπέδου, και αν φύγει ένα επίπεδο, αφαιρείται το προηγούμενο.
- **output_file**: το όνομα του αρχείου.
- **snapshot_counter**: ο αριθμός των στιγμιότυπων κατά τη μετάφραση.

### 7.2.1. Βασικές λειτουργίες της κλάσης

Η κλάση και οι μέθοδοι της εκτελούνται κατά το στάδιο μεταγλώττισης του parser και δημιουργεί στιγμιότυπα με  τα scopes entities και arguments που υπάρχουν στο πρόγραμμα τη στιγμή που κλείνει το block της κάποια συνάρτηση. Όλα αυτά τα στιγμιότυπα αποθηκεύονται σε ένα αρχείο το οποίο θα χρησιμοποιηθεί στην παραγωγή του τελικού κώδικα για την σωστή προσπέλαση των διευθύνσεων για τις μεταβλητές και τις συναρτήσεις.

Για την δημιουργία ενός νέου επιπέδου ή scope, χρησιμοποιείται η **enter_scope()**. Η συνάρτηση προσθέτει στον πίνακα των scopes ένα νέο λεξικό για τα σύμβολα, αυξάνει τον μετρητή των scopes και αρχικοποιεί των **offset_counter** στο 12 για τις νέες μεταβλητές, αφού σώσει το offset του προηγούμενου πιθανού επιπέδου για μελλοντική χρήση.

Η **exit_scope** καλείται όταν έχει τελειώσει η ανάλυση ενός επιπέδου, έτσι αν αυτό δεν είναι το μηδενικό (δηλαδή το κυρίως πρόγραμμα), καλεί την **save_table**, ώστε

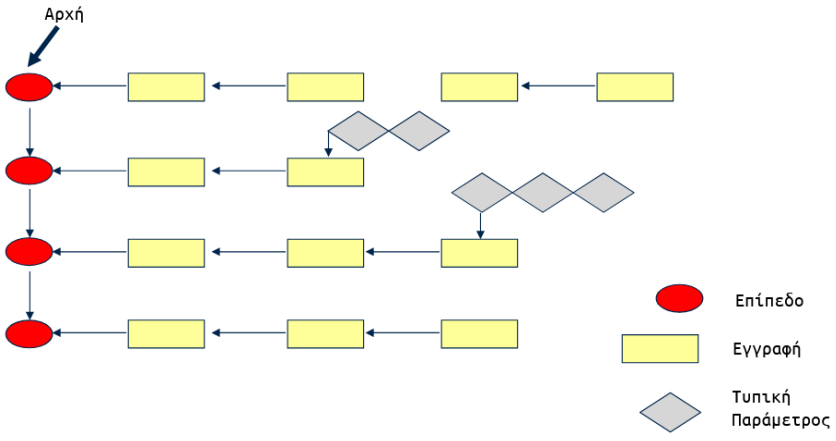
να αποθηκευτεί το στιγμιότυπο της συγκεκριμένης χρονικής στιγμής, αφαιρεί το επίπεδο από τον πίνακα και επαναφέρει το **offset** από το προηγούμενο επίπεδο, για πιθανή προσθήκη νέων μεταβλητών και τη σωστή τοποθέτηση **framelength** σε συναρτήσεις.

Η **add_symbol** είναι υπεύθυνη να καλεί τους κατάλληλους ελέγχους, οι οποίοι έχουν περιγραφθεί παραπάνω στη σημασιολογική ανάλυση και στη συνέχεια να τοποθετήσει μέσα στο λεξικό του επιπέδου, το όνομα, τον τύπο, και το offset του συμβόλου άμα είναι μεταβλητή.

Η **lookup** είναι υπεύθυνη για τον έλεγχο ύπαρξης ενός συμβόλου μέσα στον πίνακα συμβόλων.

Η **save_table** είναι η υπεύθυνη για τη δημιουργία αλλά και για την ενημέρωση του αρχείου του πίνακα συμβόλων. Κάθε φορά που καλείται προσθέτει το τωρινό στιγμιότυπο μέσα στη κλάση, δηλαδή όλα τα επίπεδα και τα σύμβολά τους. Το κάθε στιγμιότυπο αποτυπώνεται με την μορφή που φαίνεται παρακάτω, και αντιπροσωπεύει τη μορφή του πίνακα συμβόλων όπως έχουμε μάθει στη θεωρία. Το scope αποτελεί το επίπεδο, η κάθε γραμμή αποτελεί μια εγγραφή, μαζί με τις πληροφορίες της και τυπικοί παράμετροι των συναρτήσεων/διαδικασιών έχουν προστεθεί στην αντίστοιχη γραμμή τους.

```
1  Snapshot 1 begin
2  --- Scope Level 0 ---
3    α: VARIABLE, type=int, offset=12, declared at line 2
4    β: VARIABLE, type=int, offset=16, declared at line 2
5    γ: VARIABLE, type=int, offset=20, declared at line 3
6    αύξηση: FUNCTION, type=int, declared at line 4, firstquad 1, framelength=28, params: in, re
7  --- Scope Level 1 ---
8    α: PARAMETER, type=int, offset=12, mode=cv, declared at line 6
9    β: PARAMETER, type=int, offset=16, mode=ref, declared at line 7
10   T_1: TEMPORARY, type=int, offset=20
11   T_2: TEMPORARY, type=int, offset=24
12 Snapshot 1 end
```




Τέλος, η **save_framelength** είναι υπεύθυνη για την προσθήκη του **framelength** του κυρίως προγράμματος στο τέλος της συντακτικής ανάλυσης.

# 8. Παραγωγή Τελικού Κώδικα

## 8.1. Εισαγωγή

Η κλάση **RiscvCodeGenerator** είναι υπεύθυνη για τη μετατροπή του ενδιάμεσου κώδικα (τετράδων) σε κώδικα **RISC-V assembly**. Αυτό αποτελεί το τελευταίο στάδιο της μεταγλώττισης, όπου η αφηρημένη αναπαράσταση του προγράμματος μετατρέπεται σε κώδικα που μπορεί να εκτελεστεί σε πραγματικό υλικό.

## 8.2. Γενική Περιγραφή της Κλάσης

Η κλάση **RiscvCodeGenerator** λαμβάνει ως είσοδο:

- ✓ Τον ενδιάμεσο κώδικα (**λίστα τετράδων**), τον οποίο τον χρησιμοποιεί για να κάνει τα κατάλληλα **generations** της γλώσσας μηχανής **assembly**.
- ✓ Το αρχείο του πίνακα συμβόλων (**.sym**), το οποίο και διαβάζει προκειμένου να πάρει τις κατάλληλες πληροφορίες για τα framelengths
- ✓ Το όνομα του αρχείου εισόδου, το οποίο το χρησιμοποιεί για να δημιουργήσει σωστά το **.s** αρχείο **(π.χ. test.gr -> test.s)**.
- ✓ Το όνομα του κύριου προγράμματος, το οποίο χρησιμοποιείται στην δημιουργία κάποιων **labels**.

## 8.3. Βασικές λειτουργίες της κλάσης

### 8.3.1. Διαχείριση καταχωρητών και μνήμης

- **Καταχωρητές προσωρινών τιμών (t0-t6):**[4] Χρησιμοποιούνται για ενδιάμεσους υπολογισμούς και προσωρινές τιμές. Η **RiscvCodeGenerator** διατηρεί ένα σύνολο από διαθέσιμους καταχωρητές και τους αποδίδει δυναμικά σε κάθε εντολή που το απαιτεί.
- **Καταχωρητές παραμέτρων (a0-a7):** Χρησιμοποιούνται για τη μεταφορά παραμέτρων σε συναρτήσεις και για την επιστροφή τιμών.
- **Spilling**[5]**:** Όταν εξαντλούνται οι διαθέσιμοι καταχωρητές, κάποιες τιμές αποθηκεύονται προσωρινά στη στοίβα (**spill**) και επαναφέρονται όταν χρειαστεί (**restore**).
- **Αποδέσμευση καταχωρητών:** Μετά τη χρήση ενός καταχωρητή, γίνεται αποδέσμευση ώστε να μπορεί να χρησιμοποιηθεί ξανά.

[4] RISC-V Assembly Programmer's Manual - ASM Reference
[5] Lec21-Reg-alloc

### 8.3.2. Δημιουργία labels για διακλαδώσεις

- **Συσχέτιση ετικετών με τετράδες:** Κάθε τετράδα που αποτελεί στόχο διακλάδωσης (π.χ. **jump, branch**) λαμβάνει μια μοναδική ετικέτα (**L0, L1, ...,Ln**).
- **Χρήση ετικετών:** Οι εντολές διακλάδωσης (π.χ. **beq, bne, j**) χρησιμοποιούν αυτές τις ετικέτες για να μεταπηδήσουν στο σωστό σημείο του τελικού κώδικα.

### 8.3.3. Διαχείριση κλήσεων συναρτήσεων και παραμέτρων

- **Προετοιμασία στοίβας:** Πριν την κλήση, γίνεται αποθήκευση των απαραίτητων καταχωρητών και τοποθέτηση των παραμέτρων στη στοίβα.
- **Παράμετροι:** Οι παράμετροι περνιούνται είτε με τιμή (**CV**) είτε με αναφορά (**REF**) μέσω της στοίβας.
- **Access Link:** Για υποστήριξη εμφωλευμένων συναρτήσεων, τοποθετείται **access link**[6] στη στοίβα ώστε να είναι δυνατή η πρόσβαση σε μη τοπικές μεταβλητές.
- **Επιστροφή τιμής:** Η τιμή επιστρέφεται μέσω του καταχωρητή **a0** μέσω της εντολής **mv** της **assembly**.
- **Καθαρισμός στοίβας:** Μετά την επιστροφή από τη συνάρτηση, γίνεται καθαρισμός των παραμέτρων και αποκατάσταση των καταχωρητών.

### 8.3.4. Διαχείριση στοίβας για τοπικές μεταβλητές

- **Δέσμευση χώρου:** Κατά την είσοδο σε μια συνάρτηση, δεσμεύεται χώρος στη στοίβα για τοπικές μεταβλητές και προσωρινές τιμές.
- **Frame Pointer (fp):** Ο καταχωρητής **fp** δείχνει πάντα στην αρχή του τρέχοντος stack frame.
- **Αποθήκευση/φόρτωση μεταβλητών:** Οι τοπικές μεταβλητές προσπελαύνονται με βάση το offset από τον fp (π.χ. **sw t0, -8(fp)**).
- **Επιστροφή:** Κατά την έξοδο από τη συνάρτηση, γίνεται αποκατάσταση του fp και του sp και επιστροφή με **jr ra**.

## 8.4. Λειτουργικότητα και Μέθοδοι

### 8.4.1. Αρχικοποίηση και Διαχείριση Πίνακα Συμβόλων

- **parse_sym_file()**: Αναλύει το αρχείο **.sym** και δημιουργεί έναν δισδυάστατο πίνακα [στιγμιότυπα][επίπεδα], ο οποίος περιέχει λεξικά. Τα λεξικά αποτελούν τα σύμβολα (**μεταβλητές, συναρτήσεις, παράμετροι**) με

[6] Activation Records | GeeksforGeeks

τις πληροφορίες τους. Έτσι έχουμε όλο τον πίνακα συμβόλων εύκαιρο και έτοιμο για ανάλυση, αντί να πρέπει να διαβάζεται συνεχώς το αρχείο.

- **get_symbol_info(name, scope_level)**: Επιστρέφει τις πληροφορίες ενός συμβόλου από τον πίνακα, με δυνατότητα αναζήτησης σε συγκεκριμένο επίπεδο εμβέλειας.

### 8.4.2. Διαχείριση Καταχωρητών

- **get_next_temp_reg()**: Επιστρέφει τον επόμενο διαθέσιμο προσωρινό καταχωρητή (t0-t6), **με δυνατότητα spill** σε μνήμη όταν εξαντληθούν.
- **free_temp_reg(reg)**: Ελευθερώνει ένα καταχωρητή όταν δεν χρειάζεται πλέον.
- **reset_registers()**: Επαναφέρει τους δείκτες καταχωρητών μετά από κάθε συνάρτηση.

## 8.5. Μέθοδοι Παραγωγής Κώδικα

- **generate()**
  - Είναι η βασική μέθοδος που ξεκινά τη διαδικασία μετατροπής του ενδιάμεσου κώδικα (quads) σε RISC-V assembly.
  - Αρχικά, καλεί τις μεθόδους **collect_label_targets()** και **map_labels()** για να εντοπίσει και να δημιουργήσει τις απαραίτητες ετικέτες (**labels**) που θα χρησιμοποιηθούν σε διακλαδώσεις και άλματα.
  - Παράγει τον **header** του προγράμματος (π.χ. **.text, .globl**), καθώς και τις βασικές σταθερές που χρειάζονται (όπως το **newline**).
  - Για κάθε τετράδα, ελέγχει αν είναι στόχος διακλάδωσης και αν ναι, τοποθετεί την αντίστοιχη ετικέτα στον παραγόμενο κώδικα.
  - Καλεί την κατάλληλη μέθοδο παραγωγής κώδικα ανάλογα με τον τύπο της τετράδας (π.χ. **generate_assignment**, **generate_arithmetic**, **generate_comparison**, **generate_jump**, κ.λπ.).
  - Στο τέλος, αποθηκεύει τον παραγόμενο κώδικα σε αρχείο αν έχει δοθεί σχετικό όνομα εξόδου, αλλιώς παίρνει το όνομα του αρχικού αρχείου .gr που δόθηκε.
- **collect_label_targets()**
  - Διατρέχει όλες τις τετράδες και εντοπίζει ποιες από αυτές αποτελούν στόχους διακλάδωσης (δηλαδή, πού μπορεί να γίνει j**ump ή branch**).
  - Προσθέτει στη λίστα **label_targets** τα **indices** των τετράδων που πρέπει να έχουν ετικέτα, όπως:
    - Τετράδες που είναι στόχοι εντολών **jump, =, <, >, <=, >=, <>.**

        - Τετράδες που ξεκινούν ένα **block** (π.χ. **begin_block**), ώστε να μπορούν να γίνουν άλματα στην αρχή συναρτήσεων ή διαδικασιών.
    - Για το κύριο πρόγραμμα κρατάει το index της ετικέτας της main ώστε να τοποθετηθεί σωστά το entry point στην αρχή του προγράμματος.
- **map_labels()**
    - Δημιουργεί ένα **dictionary** που αντιστοιχίζει κάθε **index** τετράδας που είναι στόχος διακλάδωσης σε μια μοναδική ετικέτα (π.χ. **L0, L1, L2**), μέσω της collect_label_targets.
    - Εξασφαλίζει ότι κάθε ετικέτα είναι μοναδική και ότι το κύριο πρόγραμμα έχει τη σωστή ετικέτα ως entry point.
    - Το **dictionary** αυτό χρησιμοποιείται αργότερα από τις μεθόδους παραγωγής κώδικα για να τοποθετούν τις σωστές ετικέτες στις εντολές άλματος (**j, beq, κ.λπ**.).

## 8.6. Μέθοδοι για Συγκεκριμένες Τετράδες

- **generate_begin_block(func_name)**
    - Δημιουργεί το αρχικό τμήμα μιας συνάρτησης ή διαδικασίας.
    - Αποθηκεύει τη διεύθυνση επιστροφής (**ra**) και τον παλιό frame pointer (**fp**) στη στοίβα.
    - Θέτει τον νέο frame pointer (**fp**) και δεσμεύει χώρο για τοπικές μεταβλητές, σύμφωνα με το **framelength** της συνάρτησης.
    - Εξασφαλίζει ότι κάθε συνάρτηση ξεκινά με σωστή διαχείριση της στοίβας και των καταχωρητών.
    - Αν πρόκειται για την αρχή του κυρίως προγράμματος, δεσμεύει τον χώρο αναλόγως το framelength και μεταφέρει το stack στον global pointer (gp) για χρήση.
- **generate_end_block(func_name)**
    - Δημιουργεί το τελικό τμήμα μιας συνάρτησης ή διαδικασίας.
    - Επαναφέρει τον frame pointer (**fp**) και τη διεύθυνση επιστροφής (**ra**) από τη στοίβα.
    - Καθαρίζει το **stack frame** και εκτελεί την εντολή επιστροφής (**jr ra)**.
    - Εξασφαλίζει ότι η συνάρτηση επιστρέφει με σωστή αποκατάσταση της κατάστασης του προγράμματος.
- **generate_arithmetic(op, x, y, z)**
    - Μετατρέπει αριθμητικές πράξεις **(+, -, *, /)** σε αντίστοιχες RISC-V εντολές **(add, sub, mul, div)**.
    - Φορτώνει τα ορίσματα σε καταχωρητές, εκτελεί την πράξη και αποθηκεύει το αποτέλεσμα στη σωστή θέση (**μεταβλητή ή προσωρινή**).

- **generate_comparison(op, x, y, z)**
  - Μετατρέπει συγκρίσεις **(=, <>, <, >, <=, >=)** σε εντολές διακλάδωσης RISC-V **(beq, bne, blt, bgt, ble, bge)**.
  - Φορτώνει τα ορίσματα σε καταχωρητές και παράγει την κατάλληλη **branch** εντολή με ετικέτα στόχο.
  - Χρησιμοποιείται επίσης για την υλοποίηση συνθηκών και ελέγχου ροής.
- **generate_call(func_name, result)**
  - Χειρίζεται την κλήση συναρτήσεων ή διαδικασιών.
  - Περνάει τις παραμέτρους στη στοίβα, διαχειρίζεται το **access link** για εμφωλευμένες συναρτήσεις.
  - Αποθηκεύει την τιμή επιστροφής (δηλαδή τη διεύθυνση που βρίσκεται εκείνη τη στιγμή) και κάνει την κλήση με **jal,**.
  - Καθαρίζει τη στοίβα από παραμέτρους και **access link** μετά την επιστροφή.
  - Αν πρόκειται για συνάρτηση, περνάει την τιμή του καταχωρητή a0, στη μεταβλητή που έχει οριστεί στον πίνακα συμβόλων
- **generate_parameter(value, mode)**
  - Χειρίζεται το πέρασμα παραμέτρων σε συναρτήσεις.
    - **Για CV (με τιμή)**: Φορτώνει την τιμή και την τοποθετεί στη στοίβα.
    - **Για REF (με αναφορά)**: Υπολογίζει και περνάει τη διεύθυνση της μεταβλητής στη στοίβα.

## 8.7. Βοηθητικές Μέθοδοι

- **load_value(value, reg)**
  - Φορτώνει μια τιμή (είτε σταθερά είτε μεταβλητή) σε συγκεκριμένο καταχωρητή.
  - Αν η τιμή είναι ακέραια σταθερά, παράγει εντολή **li** για να τη φορτώσει απευθείας στον καταχωρητή.
  - Αν η τιμή είναι τοπική μεταβλητή, προσωρινή ή καθολική, παράγει εντολή **lw** με το κατάλληλο offset από το stack pointer (**sp**), ή global pointer (**gp**).
  - Αν η τιμή είναι μη-τοπική μεταβλητή, καλεί τη **generate_gnlvcode** για να βρει τη διεύθυνση και στη συνέχεια φορτώνει την τιμή από τη μνήμη.
  - Αν η μεταβλητή είναι παράμετρος με αναφορά (**ref**), φορτώνει πρώτα τη διεύθυνση και μετά την τιμή από αυτή.
- **store_value(reg, target)**

- Αποθηκεύει την τιμή που βρίσκεται σε καταχωρητή σε μια μεταβλητή (**τοπική ή μη-τοπική**).
- Για τοπικές μεταβλητές ή προσωρινές, παράγει εντολή **sw** με το κατάλληλο offset από τον **sp**.
- Για καθολικές μεταβλητές, παράγει εντολή **sw** με το κατάλληλο offset από τον **gp**.
- Για μη-τοπικές μεταβλητές, χρησιμοποιεί τη **generate_gnlvcode** για να βρει τη διεύθυνση και μετά αποθηκεύει την τιμή στη μνήμη.
- Αν ο στόχος είναι συνάρτηση (**επιστροφή τιμής**), μεταφέρει την τιμή στον καταχωρητή **a0**.

- **generate_gnlvcode(var_name)**
  - Δημιουργεί κώδικα για να υπολογίσει τη διεύθυνση μιας μη-τοπικής μεταβλητής (δηλαδή, μεταβλητής που ανήκει σε εξωτερικό scope).
  - Ξεκινά από το τρέχον frame pointer (**sp ή fp**) και ακολουθεί τα **access links** (στατική αλυσίδα) προς τα έξω, όσες φορές χρειάζεται, μέχρι να φτάσει στο σωστό **scope**.
  - Προσθέτει το offset της μεταβλητής για να βρει την τελική διεύθυνση και την αποθηκεύει στον καταχωρητή **t0**.
  - Είναι απαραίτητη για την υποστήριξη εμφωλευμένων συναρτήσεων και σωστής πρόσβασης σε μη-τοπικές μεταβλητές.
- **save_to_file()**
  1. Αποθηκεύει τον παραγόμενο RISC-V κώδικα σε αρχείο (με κατάληξη .s).
  2. Ανοίγει το αρχείο εξόδου και γράφει γραμμή-γραμμή όλες τις εντολές που έχουν παραχθεί.
  3. Ενημερώνει τον χρήστη για την επιτυχή αποθήκευση ή εμφανίζει μήνυμα λάθους αν υπάρξει πρόβλημα κατά την εγγραφή.

## 8.8 Παράδειγμα Χρήσης

Παρακάτω στις δύο εικόνες φαίνεται το αποτέλεσμα του τελικού κώδικα αν μεταγλωττίσουμε το αρχείο παράδειγμα.gr:

```
πρόγραμμα Παράδειγμα
δήλωση x, y
αρχή_προγράμματος
    x := 10;
    y := x + 5;
    γράψε y
τέλος_προγράμματος
```

Η αριστερά φωτογραφία δείχνει το παραγόμενο αρχείο παράδειγμα.s ενώ η δεξιά δείχνει το αποτέλεσμα του τελικού κώδικα αν ο προγραμματιστής δώσει την εντολή **python3 greek_5387_5388 παράδειγμα.gr --debug** μέσω του τερματικού.

```
.text
.globl L0

.data
str_nl: .asciz "\n"

.text

j L0


L0:

addi sp, sp, 24
mv gp, sp
li t0, 10
sw t0, -12(gp)
lw t0, -12(gp)
li t1, 5
add t2, t0, t1
sw t2, -20(gp)
lw t0, -20(gp)
sw t0, -16(gp)
lw t0, -16(gp)
mv a0, t0
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall
li a0, 0
li a7, 93
ecall
```

```
[DEBUG] Generated RISC-V code:
.text
.globl L0

.data
str_nl: .asciz "\n"

.text

j L0


L0:

addi sp, sp, 24
mv gp, sp
li t0, 10
sw t0, -12(gp)
lw t0, -12(gp)
li t1, 5
add t2, t0, t1
sw t2, -20(gp)
lw t0, -20(gp)
sw t0, -16(gp)
lw t0, -16(gp)
mv a0, t0
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall
li a0, 0
li a7, 93
ecall
```

# 9. Προβλήματα και Μελλοντικές Βελτιώσεις

## 9.1. Προβλήματα

I. Η βελτιστοποίηση **constant folding** προκαλεί προβλήματα στη διαχείριση των καταχωρητών στον τελικό κώδικα RISC-V, όπως και αναφέρθηκε προηγουμένως, για αυτό και τελικά δεν χρησιμοποιείται.

II. Ο έλεγχος αρχικοποίησης μεταβλητών μπορεί να βελτιωθεί για να λαμβάνει υπόψη όλες τις πιθανές διαδρομές εκτέλεσης.

III. Υπάρχουν κάποια edge cases τα οποία δεν χειριστήκαμε σωστά και έτσι δεν εκτυπώνεται μήνυμα σφάλματος στο 100% των περιπτώσεων. Βέβαια τα αρχεία που δημιουργούνται είναι κενά (πράγμα που υποδηλώνει σφάλμα στην μεταγλώττιση).

IV. Υπάρχουν προβλήματα με τις παραμέτρους στις συναρτήσεις και κυρίως σε κάποια sw με τα offset.

## 9.2. Μελλοντικές Βελτιώσεις

- ✓ Υλοποίηση περισσότερων τεχνικών βελτιστοποίησης (π.χ., **dead code elimination**) και παροχή βελτιώσεων όχι μόνο για το στάδιο του ενδιάμεσου κώδικα αλλά και του τελικού.
- ✓ Προσθήκη δυναμικού αριθμού παραμέτρων μέσα σε μια συνάρτηση, τύπου σαν τα **variadic functions**[7] της C.
- ✓ Επέκταση των τύπων μηνυμάτων σφάλματος, με καλύτερη πρόβλεψη και προειδοποιήσεις
- ✓ Καλύτερη διαχείριση των registers στην παραγωγή του τελικού κώδικα μέσω κάποιων τεχνικών όπως **Coalescing, Graph Coloring Register Allocation.**

# 10. Βιβλιογραφία

[1] Γ. Μανής and G. Manis, "Εγχειρίδιο Σχεδίασης και Ανάπτυξης Μεταγλωττιστών," Nov. 10, 2023. https://repository.kallipos.gr/handle/11419/11371

*[2] ΣΧΕΔΙΑΣΗ ΚΑΙ ΚΑΤΑΣΚΕΥΗ ΜΕΤΑΓΛΩΤΤΙΣΤΩΝ - Πανεπιστημιακές Εκδόσεις Κρήτης*. 2024. [Online]. Available: https://cup.gr/book/schediasi-ke-kataskevi-metaglottiston/

[7] Variadic Functions in C | GeeksforGeeks